SET-A

1.
```
import pandas as pd
from sklearn.datasets import load_diabetes

# Load dataset
diabetes = load_diabetes()

# Create DataFrame
df = pd.DataFrame(data=diabetes.data,
columns=diabetes.feature_names)

# Add target column
df['target'] = diabetes.target

#Display First 5 Rows
print(df.head())

#Display Dataset Shape
print("Shape of dataset:", df.shape)

#Check Missing Values
print(df.isnull().sum())

#Fill with Mean
df.fillna(df.mean(), inplace=True)

#Drop Missing Rows
df.dropna(inplace=True)
```

2

```
 import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_diabetes
```

```
#Load Dataset
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target



# First 5 rows
print(df.head())

# Dataset info
print(df.info())

# Statistical summary
print(df.describe())

# Check missing values
print(df.isnull().sum())

#Correlation Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()

#Scatter Plot (BMI vs Target)
plt.figure(figsize=(6, 4))
plt.scatter(df['bmi'], df['target'])
plt.xlabel("BMI")
plt.ylabel("Target (Disease Progression)")
```

```
plt.title("BMI vs Target")
plt.show()

#Histogram of Age Feature
plt.figure(figsize=(6, 4))
plt.hist(df['age'], bins=20)
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.title("Age Distribution")
plt.show()
```

3

```
# ============================
# 1. Import Libraries
# ============================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# ============================
# 2. Load Dataset
# ============================
diabetes = load_diabetes()
```

```
df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
df['target'] = diabetes.target

# ============================
# 3. Basic EDA
# ============================
print("First 5 rows:\n", df.head())
print("\nShape:", df.shape)
print("\nMissing values:\n", df.isnull().sum())
print("\nStatistical Summary:\n", df.describe())

# ============================
# 4. Correlation Heatmap
# ============================
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()

# ============================
# 5. Scatter Plot (BMI vs Target)
# ============================
plt.figure(figsize=(6, 4))
plt.scatter(df['bmi'], df['target'])
plt.xlabel("BMI")
plt.ylabel("Target")
plt.title("BMI vs Target")
plt.show()
```

```
# ============================
# 6. Histogram of Age
# ============================
plt.figure(figsize=(6, 4))
plt.hist(df['age'], bins=20)
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.title("Age Distribution")
plt.show()

# ============================
# 7. Separate Features & Target
# ============================
X = df.drop('target', axis=1)
y = df['target']

# ============================
# 8. Train-Test Split
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ============================
# 9. Train Linear Regression Model
# ============================
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# ============================
# 10. Prediction
# ============================
y_pred = model.predict(X_test)

# ============================
# 11. Evaluation
# ============================
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation:")
print("RMSE:", rmse)
print("R² Score:", r2)
```

4.a

Unsupervised Learning

Unsupervised learning is a type of machine learning in which the model is trained on **unlabeled data**. Unlike supervised learning, there is **no target/output variable**. The goal is to discover **hidden patterns, structures, or relationships** in the data.

🔍 Types of Unsupervised Learning (Detailed Explanation)

🔹 1. Clustering

📌 Definition:

Clustering is the process of grouping similar data points into clusters such that:

- Data points in the **same cluster are similar**
- Data points in **different clusters are dissimilar**

Common Algorithms:

- K-Means Clustering
- Hierarchical Clustering
- DBSCAN

⚙ **Working (K-Means Example):**

1. Choose number of clusters (K)
2. Randomly assign centroids
3. Assign data points to nearest centroid
4. Update centroids
5. Repeat until convergence

Example:

Customer segmentation:

- Group customers based on purchase behavior
- Clusters: High spenders, medium, low

Applications:

- Market segmentation
- Image segmentation
- Document grouping

---

2. Association Rule Learning

Definition:

It discovers **interesting relationships (rules)** between variables in large datasets.

Common Algorithms:

- Apriori Algorithm
- FP-Growth Algorithm

🔑 Key Concepts:

- **Support** → Frequency of itemset
- **Confidence** → Likelihood of rule
- **Lift** → Strength of association

📌 Example:

Market Basket Analysis:

- Rule: {Milk → Bread}
- Means customers buying milk also buy bread

🚀 Applications:

- Product recommendation
- Cross-selling
- Retail analysis

🔹 3. Dimensionality Reduction

📖 Definition:

Reduces the number of input variables (features) while keeping important information.

🔍 Common Algorithms:

- PCA (Principal Component Analysis)
- t-SNE
- LDA

⚙️ Working (PCA):

- Converts correlated variables into **uncorrelated principal components**
- Keeps components with maximum variance

📌 Example:

- Reducing 100 features into 2D for visualization

🚀 Applications:

- Data visualization
- Noise reduction

- Faster computation

🔹 4. Anomaly Detection

📌 Definition:

Identifies rare or unusual data points that do not follow normal patterns.

🔧 Techniques:

- Isolation Forest
- One-Class SVM
- Statistical methods

⚙️ Working:

- Model learns normal data behavior
- Points far from normal pattern are flagged as anomalies

📊 Example:

- Fraud detection in banking
- Detect unusual transactions

🚀 Applications:

- Fraud detection
- Intrusion detection
- Fault detection

🔹 5. Density-Based Learning (Density Estimation)

📌 Definition:

It estimates how data points are distributed in space.

🔧 Common Methods:

- Kernel Density Estimation (KDE)
- Gaussian Mixture Models (GMM)

⚙️ Working:

- Finds regions of **high data density**
- Helps identify clusters and patterns

 Example:

- Estimating distribution of population age

 Applications:

- Pattern recognition
- Data distribution modeling
- Image processing

SET-B

1.

```
import pandas as pd
from sklearn.datasets import load_breast_cancer

# Load dataset
data = load_breast_cancer()

# Create DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
```

```
# Add target column
df['target'] = data.target

# Display dataset structure
print("Dataset Info:\n")
print(df.info())

print("\nStatistical Summary:\n")
print(df.describe())

# Check missing values
print("\nMissing Values:\n")
print(df.isnull().sum())

# Handle missing values (if any)
df.fillna(df.mean(), inplace=True)

# Final check
print("\nMissing Values After Handling:\n")
print(df.isnull().sum())
```

2.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

```
# ============================
# 1. Load Dataset
# ============================
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# ============================
# 2. Basic EDA
# ============================
print("First 5 rows:\n", df.head())
print("\nDataset Info:\n")
print(df.info())
print("\nStatistical Summary:\n", df.describe())
print("\nMissing Values:\n", df.isnull().sum())

# ============================
# 3. Distribution Plots (for few features)
# ============================
features = ['mean radius', 'mean texture', 'mean perimeter', 'mean area']

for feature in features:
    plt.figure()
    sns.histplot(df[feature], kde=True)
    plt.title(f"Distribution of {feature}")
    plt.xlabel(feature)
    plt.ylabel("Frequency")
    plt.show()
```

```
# ============================
# 4. Histogram of Mean Radius
# ============================
plt.figure()
plt.hist(df['mean radius'], bins=20)
plt.title("Histogram of Mean Radius")
plt.xlabel("Mean Radius")
plt.ylabel("Frequency")
plt.show()

# ============================
# 5. Correlation Heatmap
# ============================
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()

# ============================
# 6. Train-Test Split
# ============================
X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print("\nTraining set shape:", X_train.shape)
```

```
print("Testing set shape:", X_test.shape)
```

3.

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# ============================
# 1. Load Dataset
# ============================
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# ============================
# 2. Separate Features & Target
# ============================
X = df.drop('target', axis=1)
y = df['target']

# ============================
# 3. Train-Test Split
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
# ============================
# 4. Apply Decision Tree Classifier
# ============================
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# ============================
# 5. Predict Test Data
# ============================
y_pred = model.predict(X_test)

# ============================
# 6. Display Predictions
# ============================
print("Predicted values:\n", y_pred)
print("\nActual values:\n", y_test.values)
```

4.b

wo Methods to Reduce Overfitting

 1. Regularization

 Definition:

Regularization adds a **penalty term** to the model to prevent it from becoming too complex.

 Types:

- **L1 Regularization (Lasso)**
- **L2 Regularization (Ridge)**

⚙ **How it Works:**

- Penalizes large coefficients
- Forces the model to keep weights small
- Reduces model complexity

 Example:

In Linear Regression:

- Instead of minimizing only error, we add a penalty term
- Helps avoid fitting noise in data

 Result:

 Simpler model with better generalization

---

 2. Cross-Validation

 Definition:

Cross-validation splits data into multiple parts to evaluate model performance more reliably.

⚙ **How it Works (K-Fold CV):**

1. Split data into K parts (folds)
2. Train on K-1 folds
3. Test on remaining fold
4. Repeat K times
5. Average performance

 Example:

- 5-fold cross-validation for model evaluation

 Result:

 Ensures model performs well on unseen data
 Prevents overfitting to a single training set

SET-C

1.

```
import pandas as pd
from sklearn.datasets import load_wine

# ============================
# 1. Load Dataset
# ============================
data = load_wine()

# Create DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)

# Add target column
df['target'] = data.target

# ============================
# 2. Display Column Names
```

```
# ============================
print("Column Names:\n", df.columns)

# ============================
# 3. Summary Statistics
# ============================
print("\nSummary Statistics:\n", df.describe())

# ============================
# 4. Separate Features & Target
# ============================
X = df.drop('target', axis=1)
y = df['target']

print("\nFeatures (X) shape:", X.shape)
print("Target (y) shape:", y.shape)
```

2.

```
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# ============================
# 1. Load Dataset
# ============================
```

```
data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# ============================
# 2. Separate Features & Target
# ============================
X = df.drop('target', axis=1)
y = df['target']

# ============================
# 3. Apply StandardScaler
# ============================
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ============================
# 4. Train-Test Split
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# ============================
# 5. Apply Logistic Regression
# ============================
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
```

```
# ============================
# 6. Prediction
# ============================
y_pred = model.predict(X_test)

# ============================
# 7. Output
# ============================
print("Predicted values:\n", y_pred)
print("\nActual values:\n", y_test.values)
```

3.

```
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# ============================
# 1. Load Dataset
# ============================
data = load_wine()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# ============================
# 2. Separate Features & Target
# ============================
```

```
X = df.drop('target', axis=1)
y = df['target']

# ============================
# 3. Train-Test Split (same split for fair comparison)
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ============================
# 4. WITHOUT SCALING
# ============================
model1 = LogisticRegression(max_iter=1000)
model1.fit(X_train, y_train)

y_pred1 = model1.predict(X_test)
acc1 = accuracy_score(y_test, y_pred1)

print("Accuracy WITHOUT Scaling:", acc1)

# ============================
# 5. WITH SCALING
# ============================
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
model2 = LogisticRegression(max_iter=1000)
model2.fit(X_train_scaled, y_train)

y_pred2 = model2.predict(X_test_scaled)
acc2 = accuracy_score(y_test, y_pred2)

print("Accuracy WITH Scaling:", acc2)

# ============================
# 6. Comparison
# ============================
print("\nComparison:")
print("Without Scaling:", acc1)
print("With Scaling   :", acc2)
```

4.b

1. L1 Regularization (Lasso)

 Definition:

L1 adds the **absolute values of coefficients** as a penalty term to the loss function.

$Loss = RSS + \lambda \sum |\beta_i|$

⚙ **Key Characteristics:**

- Can shrink some coefficients to **exactly zero**
- Performs **feature selection automatically**

 Role:

 Removes irrelevant features and simplifies the model

 Example:

- In a dataset with many features, L1 will keep only the most important ones

---

🔹 2. L2 Regularization (Ridge)

📌 Definition:

L2 adds the **square of coefficients** as a penalty.

$$Loss = RSS + \lambda \sum \beta_i^2$$

⚙️ Key Characteristics:

- Reduces magnitude of coefficients
- Does **not make them exactly zero**

🎯 Role:

✅ Controls model complexity and prevents large weights

🧠 Example:

- Useful when all features are important but need regularization

---

🔹 3. Elastic Net Regularization

📌 Definition:

Combination of **L1 and L2 regularization**

$$Loss = RSS + \lambda_1 \sum |\beta_i| + \lambda_2 \sum \beta_i^2$$

⚙️ Key Characteristics:

- Combines benefits of both L1 and L2
- Handles correlated features better

🎯 Role:

✅ Balances feature selection and coefficient shrinkage

🧠 Example:

- Used when dataset has many correlated variables

SET-4

1.

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

# ============================
# 1. Load Iris Dataset
# ============================
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# ============================
# 2. Apply StandardScaler
# ============================
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Convert scaled data to DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=iris.feature_names)
# ============================
# 3. Display Output
```

```
# ============================
print("Original Data (First 5 rows):\n", df.head())

print("\nScaled Data (First 5 rows):\n", scaled_df.head())
```

2.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# ============================
# 1. Load Dataset
# ============================
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Add target column
df['target'] = iris.target

# ============================
# 2. Check Missing Values
# ============================
print("Missing Values:\n", df.isnull().sum())

# ============================
# 3. Pairplot
# ============================
sns.pairplot(df, hue='target')
plt.show()
```

```
# ============================
# 4. Scatter Plot (Sepal Length vs Petal Length)
# ============================
plt.figure()
sns.scatterplot(
    x='sepal length (cm)',
    y='petal length (cm)',
    hue='target',
    data=df
)
plt.title("Sepal Length vs Petal Length")
plt.show()
```

3.

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# ============================
# 1. Load Dataset
# ============================
iris = load_iris()
```

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target

# ============================
# 2. Separate Features & Target
# ============================
X = df.drop('target', axis=1)
y = df['target']

# ============================
# 3. Train-Test Split (80:20)
# ============================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ============================
# 4. Feature Scaling (important for KNN)
# ============================
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ============================
# 5. KNN Model (k = 5)
# ============================
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
```

```
y_pred_knn = knn.predict(X_test_scaled)
knn_acc = accuracy_score(y_test, y_pred_knn)

# ============================
# 6. Logistic Regression Model
# ============================
lr = LogisticRegression(max_iter=200)
lr.fit(X_train_scaled, y_train)

y_pred_lr = lr.predict(X_test_scaled)
lr_acc = accuracy_score(y_test, y_pred_lr)

# ============================
# 7. Compare Accuracy
# ============================
print("KNN Accuracy (k=5):", knn_acc)
print("Logistic Regression Accuracy:", lr_acc)
```

4.

Support Vector Machine (SVM)

SVM is a supervised learning algorithm used for **classification and regression**. It finds the **optimal hyperplane** that separates data into different classes with maximum margin.

□ 1. Hard Margin SVM

□ Definition:

Hard Margin SVM is used when the dataset is **perfectly linearly separable** (no overlap between classes).

🎯 Objective:

Find a hyperplane that separates classes with **maximum margin** and **no misclassification**.

📐 Equation of Hyperplane:

$$w \cdot x + b = 0$$

📏 Optimization Objective:

$$min \frac{1}{2} ||w||^2$$

Subject to:
$$y_i(w \cdot x_i + b) \geq 1$$

⚙️ Key Points:

- No tolerance for misclassification
- Works only when data is perfectly separable
- Sensitive to noise and outliers

📌 Example:

- Clean dataset where two classes are completely separable

⚠️ Limitation:

❌ Not practical for real-world noisy data

🔹 2. Soft Margin SVM

📘 Definition:

Soft Margin SVM allows **some misclassification** to handle noisy or overlapping data.

🎯 Objective:

Maximize margin while allowing some errors

📐 Optimization Equation:

$$min\ \frac{1}{2}||w||^2 + C\sum \xi_i$$

📏 Constraint:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

⚙️ Key Terms:

- **ξᵢ (Slack Variable)** → allows violations
- **C (Regularization parameter)** → controls trade-off:
  - Large C → less error, smaller margin
  - Small C → more error, larger margin